

SYSTEM AND METHOD FOR IMPLEMENTING
APPLICATION FUNCTIONALITY WITHIN A
NETWORK INFRASTRUCTURE

BACKGROUND OF THE INVENTION

5 **1. Related Applications.**

The present invention claims priority from U.S. Provisional Patent Application No. 60/197,490 entitled CONDUCTOR GATEWAY filed on April 17, 2000.

10 **2. Field of the Invention.**

The present invention relates, in general, to network information access and, more particularly, to software, systems and methods for implementing application-independent functionality within a network infrastructure.

15 **3. Relevant Background.**

Increasingly, business data processing systems, entertainment systems, and personal communications systems are implemented by computers across networks that are interconnected by internetworks (e.g., the Internet). The Internet is rapidly emerging as the preferred system for distributing and exchanging data. Data exchanges support applications including electronic commerce, broadcast and multicast messaging, videoconferencing, gaming, and the like.

25 The Internet is a collection of disparate computers and networks coupled together by a web of interconnections using standardized communications protocols. The Internet

is characterized by its vast reach as a result of its wide and increasing availability and easy access protocols. Unfortunately, the heterogeneous nature of the Internet makes it difficult for the hardware and software that
5 implement the Internet to add functionality.

The Open System Interconnection (OSI) network model usefully describes networked data communication, such as the Internet, as a series of logical layers or protocol layers. Each layer provides services to the layer above it, and shields the layer above it from details of lower layers. Each layer is configured to communicate with other similar level layers. In general, computers at network nodes (e.g., clients and servers) implement higher level processes including application layer, presentation layer, and session layer processes. Lower level processes, including network layer, data link layer and physical layer operate to place data in a form suitable for communication across a raw communication channel or physical link. Between the higher and lower level
10 processes is a transport layer that typically executes on a machine at the network node, but is highly dependent on the lower level processes.
15
20

While standards exist for these layers, application designers have a high level of control and can implement semantics and functionality at the higher layers with a great deal of latitude. In contrast, lower layers are highly standardized. Implementing or modifying functionality in a lower layer protocol is very difficult as such changes can affect almost all users of the network. Devices such as routers that are typically associated with infrastructure operate exclusively at the lower protocol layers making it difficult or impossible to
25
30 implement functionality such as real-time processing, data

compression, encryption and error correction within a network infrastructure.

Although the term "Internet infrastructure" encompasses a variety of hardware and software mechanisms, the term primarily refers to routers, router software, and physical links between these routers that function to transport data packets from one network node to another.

Internet infrastructure components such as routers and switches are, by design, asynchronous. Also by design, it is difficult to accurately predict or control the route a particular packet will take through the Internet. This architecture is intended to make the Internet more robust in the event of failures, and to reduce the cost, complexity and management concerns associated with infrastructure components. As a result, however, a particular node or machine cannot predict the capabilities of the downstream mechanisms that it must rely on to deliver a packet to its destination. A sending node cannot expect all mechanisms in the infrastructure to support the functions and/or syntax necessary to implement such functions as real time processing, data compression, encryption, and error correction.

For example, it is difficult if not impossible to conduct synchronous or time-aware operations over the Internet. Such operations include, for example, real-time media delivery, access to financial markets, interactive events, and the like. While each IP packet includes information about the time it was sent, the time base is not synchronous between sender and receiver, making the time indication inaccurate. Packets are buffered at various locations through the Internet infrastructure, and there is no accurate way to ascertain the actual age or

time of issue of the packet. Hence, critical packets may arrive too late.

Data compression is a well-known technique to improve the efficiency of data transport over a communication link. Typically, data compression is performed at nodes sending the data and decompression performed at a node receiving the data. Infrastructure components responsible for sending the information between the sending and receiving processes do not analyze whether effective compression has been performed, nor can the infrastructure implement compression on its own. Where either the sending or receiving process is incapable of effective compression, the data goes uncompressed. This creates undesirable burden that affects all users. While modems connecting a user over a phone line often apply compression to that link, there is no analogous function within the Internet infrastructure itself. A need exists for Internet infrastructure components that compress data between network nodes to improve transport within the Internet.

Similarly, encryption and other data security techniques are well known techniques to ensure only authorized users can read data. Like compression, however, encryption is typically performed by user-level and application-level processes. If either sending or receiving process cannot perform compatible encryption, the data must be sent in the clear or by non-network processes. A need exists for Internet infrastructure components that apply encryption or other security processes transparently to users.

As another example, forward error correction (FEC) is a known technique to reduced traffic volume, reduce latency, and/or increase data transfer speed over lossy

connections. FEC adds redundant information, also referred to as error correction code, to the original message, allowing the receiver to retrieve the message even if it contains erroneous bits. FEC coding can enhances decoded bit error rate values three order of magnitude relative to systems not implementing any FEC techniques. When the error can be detected and corrected at the receiving end, there is less need to resend data. FEC is extensively used in many digital communication systems at some level and in mass storage technology to compensate for media and storage system errors.

However, FEC is not used within the Internet infrastructure. This stems in part from the additional complexity, cost and management tasks that such capability would impose on the system hardware and software. FEC requires that the sender and receiver both implement compatible FEC processes. Hence, most if not all infrastructure components would have to be replaced or modified to implement FEC in an effective manner. Efforts to implement FEC between sending and receiving nodes are outlined in IETF RFC 2733. This proposed standard applies to real time transport protocol (RTP) communications between a client and server. This FEC method affects endpoints to a data transfer, but does not affect servers and or other infrastructure components located between the endpoints. Hence, a need exists for systems and methods that implement FEC within the Internet infrastructure to offer the benefits of FEC technology seamlessly to network users.

In most cases these types of functionality are implemented in higher level processes (e.g., the OSI application layer, presentation layer, session layer and/or transport layer). However this requires that

sending and receiving nodes implement a common syntax. For example, both sending and receiving nodes must implement complementary encryption/decryption processes, however once this is ensured, the communication will be encrypted through out transport. In practice there are multiple standards for real-time processing, encryption, compression, and error correction, and one or the other node may be unable to support the protocols of the other nodes. Hence, it is desirable to implement such functionality in a manner that is independent of the higher level processes so that otherwise incompatible or incapable application-level processes can benefit.

In other cases, for example real time processing and error correction, it is desirable to have the functionality implemented within the network infrastructure, not only between the nodes. For example, implementing error correction only between the sending and receiving nodes is only a partial solution, as the infrastructure components that operate at lower network layers (e.g., transport, network, data link and/or physical layer) cannot read error correction codes inserted at higher network layers. As another example, traffic prioritization within the network benefits from knowledge of when packets were actually sent so that they can be delivered in time for real-time processes.

A particular need exists in environments that involve multiple users accessing a network resource such as a web server. Web servers are typically implemented with rich functionality and are often extensible in that the functionality provided can be increased modularly to provide general-purpose and special-purpose functions. Examples include information services, broadcast, multicast and videoconference services, as well as most

electronic commerce (e-commerce) applications. In these applications it is important that functionality provided by network-connected resources be provided in a dependable, timely and efficient manner.

5 Many e-commerce transactions are abandoned by the user because system performance degradations frustrate the purchaser before the transaction is consummated. While a transaction that is abandoned while a customer is merely browsing through a catalog may be tolerable, abandonment
10 when the customer is just a few clicks away from a purchase is highly undesirable. However, existing Internet transport mechanisms and systems do not allow the e-commerce site owner any ability to distinguish between the "just browsing" and the "about to buy" customers as
15 this information is represented at higher network layers that are not recognized by the infrastructure components. In fact, the vagaries of the Internet may lead to the casual browser receiving a higher quality of service while the about-to-buy customer becomes frustrated and abandons
20 the transaction.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a system for implementing functionality within a network on behalf of first and second computers communicating with
25 each other through the network. A front-end computer is provided within the network having an interface for communicating data traffic with the first computer. A back-end computer is also implemented within the network having an interface for communicating data traffic with
30 the second computer. A communication channel couples the front-end computer and the back-end computer. Data traffic is encoded over the communication channel in a

first process in the front-end computer. Data traffic is also encoded over the communication channel in a second process in the back-end computer, wherein the first process and the second process implement compatible semantics.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

FIG. 2 shows in block-diagram form entity relationships in a system in accordance with the present invention;

FIG. 3 shows a domain name system used in an implementation of the present invention;

FIG. 4 shows front-end components of FIG. 2 in greater detail;

FIG. 5 shows back-end components of FIG. 2 in greater detail;

FIG. 6 illustrates in flow-diagram form processes involved in an exemplary implementation of the present invention;

FIG. 7 shows a conceptual block diagram of particular components introduced in FIG. 2 in greater detail;

FIG. 8 shows exemplary pre-processing processes; and

FIG. 9 illustrates exemplary post-processing processes.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present inventions involve improvements to communication channels implemented through a public network such as the Internet. These improvement are
5 enabled by using front-end and back-end servers, typically implemented as web servers, that are located within the network. It is difficult to define a clear demarcation point for what mechanisms are "in the network" in contrast with mechanisms "outside of the network". Typically,
10 devices outside the network, such as clients and servers, establish a channel through the network with each other. Using the OSI network model referenced above, all of the software and hardware mechanisms below the "network" protocol layer in the OSI model in the client and server
15 computers can be considered within the network. Similarly processes and mechanisms that operate above the network level in the client and server can be considered "outside the network".

Given the terminology framework above, certain
20 features of the present invention involve implementing processes that provide higher-layer services within the network. For example, services typically associated with the "presentation layer" or "application layer" such as compression and encryption are implemented within the
25 network. In accordance with the present invention, these higher-layer processes are implemented between machines within the network in a manner that is preferably transparent to the computers outside the network. In this manner, so long as a common semantic is defined for a pair
30 or set of machines within the network, it is not necessary to modify clients, servers, or other infrastructure components such as routers to recognize the semantic used to provide these higher-layer functions.

A first set of inventions relate to the improved functionality and metrics available when cooperating front-end and back-end servers are used to transport data through the public network. This first class of inventions enable an enhanced communication channel in which both ends can be synchronized and so easily know when the other end performed specific operations such as datagram generation and transmission. This enables each side to take actions based on the knowledge that was previously only available to the transmitting side. Other functionality includes compression of traffic between front-end and back-end using public or proprietary compression mechanisms that can be readily selected and optimized for the particular content data currently being transported. Similarly, encryption/decryption mechanisms can be employed between the front-end and back-end for enhanced security without impacting either a web server or web client that are principles of the transaction. Forward error correction can be used to reduce the quantity of traffic, improve latency, and/or increase speed of the transport between front-end and back-end components.

A second set of inventions relates to performance and functionality improvements enabled by implementing the front-end and back-end computers as dynamically re-configurable elements. This second class of inventions enables multiple front-ends to connect with and service multiple back-ends and/or one or more web servers or web sites. These inventions also include the ability for one front-end to service multiple back-ends and by extension multiple web servers or web sites. Similarly, one front-end can service multiple web servers or content providers directly.

In one aspect, the present invention involves a system for multiplexing data from a plurality of links or channels onto a shared bandwidth channel. The plurality of links may be fixed-bandwidth links, or may themselves be shared bandwidth links. The plurality of links may comprise a homogenous user-level protocol, such as HTTP, or may comprise a variety of user level protocols such as HTTP, FTP, NNTP, SMTP and the like. The plurality of links may similarly comprise homogenous network-layer and/or physical layer protocols, or may comprise a varied set of network-layer and physical layer protocols.

The shared bandwidth channel allows a variety of services to be provided. Some advantages are achieved simply by multiplexing multiple links onto a single channel. This combination enables the single channel to be persistent thereby avoiding overhead associated with setting up, maintaining and breaking down connections that would otherwise be required of each the multiple links. The single shared channel can also include more information than the protocols of the plurality of links allow such as time synchronization information and quality of service information.

In a particular embodiment, the shared bandwidth channel transports packets that are composed by selecting data from the plurality of links in an order and rate determined to provide differential levels of service between packets. The differential service levels may mean that some of the data are transported with lower latency and/or higher quality of service than other data. The criteria for providing differential levels of service is not limited, but in particular embodiments is based on content type, user identity, user history, and session statistics.

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments, including wireless network environments, as well as small network environments such as conventional LAN systems.

The present invention is particularly useful in applications where there is an large amount of data communicated between web servers and web clients (i.e., browser software) or where timeliness (e.g., low latency transport) is important. For example, real-time stock quotes, multi-player games, multi-tiered service to ASP (application service provider) software distribution models benefit from the improvements provided by the present invention. Although the present invention will be described in terms of particular applications, these examples are provided to enhance understanding and are not a limitation of the essential teachings of the present invention.

For purposes of this document, a web server is a computer running server software coupled to the World Wide Web (i.e., "the web") that delivers or serves web pages. The web server has a unique IP address and accepts connections in order to service requests by sending back responses. A web server differs from a proxy server or a gateway server in that a web server has resident a set of resources (i.e., software programs, data storage capacity, and/or hardware) that enable it to execute programs to

provide an extensible range of functionality such as generating web pages, accessing remote network resources, analyzing contents of packets, reformatting request/response traffic and the like using the resident resources.

In contrast, a proxy simply forwards request/response traffic on behalf of a client to resources that reside elsewhere, or obtains resources from a local cache if implemented. A web server in accordance with the present invention may reference external resources of the same or different type as the services requested by a user, and reformat and augment what is provided by the external resources in its response to the user. Commercially available web server software includes Microsoft Internet Information Server (IIS), Netscape Netsite, Apache, among others. Alternatively, a web site may be implemented with custom or semi-custom software that supports HTTP traffic.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented. Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token Ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, or peer-to-peer type network. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another

network mechanism such as a fibre channel fabric or conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Microsoft Windows, MacIntosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, EPOCOS, and the like. Moreover, the present invention is readily extended to

network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

The present invention operates in a manner akin to a private network 200 implemented within the Internet infrastructure as shown in FIG. 2. Private network 200 enhances communications between a client 205 and a web site 210 by implementing any of a variety of processes that enhance efficiency and/or functionality independently of client 205 and/or server 210. These processes include time synchronization processes, quality of service management processes, compression processes, security processes, and error correction processes.

In the specific examples herein client 205 comprises a network-enabled graphical user interface such as a web browser. However, the present invention is readily extended to client software other than conventional web browser software. Any client application that can access a standard or proprietary user level protocol for network

access is a suitable equivalent. Examples include client applications for file transfer protocol (FTP) services, voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may access a network application such as a database management system (DBMS) in which case the client application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client application may communicate via a wireless application protocol or the like.

For convenience, the term "web site" is used interchangeably with "web server" in the description herein although it should be understood that a web site comprises a collection of content, programs and processes implemented on one or more web servers. A web site is owned by the content provider such as an e-commerce vendor whereas a web server refers to set of programs running on one or more machines coupled to an Internet node. The web site 210 may be hosted on the site owner's own web server, or hosted on a web server owned by a third party. A web hosting center is an entity that implements one or more web sites on one or more web servers using shared hardware and software resources across the multiple web sites. In a typical web infrastructure, there are many web browsers, each of which has a TCP connection to the web server in which a particular web site is implemented. The present invention adds two components to the infrastructure: a front-end 201 and back-end 203. Front-end 201 and back-end 203 are coupled by a managed data communication link 202 that forms, in essence, a private network.

Front-end mechanism 201 serves as an access point for client-side communications. In the process of translating a requested domain name into an IP address of a particular server hosting the requested domain name, mechanisms described in reference to FIG. 3 operate to select a particular front-end mechanism 201. In effect, the domain is dynamically assigned to the selected front-end mechanism. More than one front-end 201 may host a single domain. So long as a client 205 associates the domain name with the IP address of the selected front-end 201, all client requests to the domain will be routed to the selected front-end 201.

Front-end mechanism 201 implements a set of processes in the dynamically assigned domain that implement a gateway that functions as a substitute for the web server(s) implementing web site 210 (i.e., from the perspective of client 205, front-end 201 appears to be the web site 210). Front-end 201 comprises, for example, a computer that sits "close" to clients 205. By "close", it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and a web site 210. Desirably, front-end computers have as fast a connection as possible to the clients 205. For example, the fastest available connection may be implemented in a point of presence (POP) of an Internet service provider (ISP) 106 used by a particular client 205. However, the placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications it is more practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance between front-end 201 and clients 205 may be desirable in some applications as this distance will allow for

selection amongst a greater number front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when particular routes and/or front-ends become congested or otherwise unavailable.

Transport mechanism 202 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from web site 210. Transport mechanism 202 communicates data packets using a proprietary protocol called transport morphing protocol™ or TMP™. Transport morphing protocol and TMP are trademarks or registered trademarks of Circadence Corporation in the United States and other countries. TMP is implemented over the public Internet infrastructure in the particular example. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general purpose network 101. Unlike the general purpose Internet, front-end 201 and back-end 203 are programmably assigned to serve accesses to a particular web site 210 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the web site owner. The present invention implements a many-to-many mapping of front-ends to back-ends. Because the front-end to back-end mappings can be dynamically changed, a fixed hardware infrastructure can be logically reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

Front-end 201 together with back-end 203 function to reduce traffic across the TMP link 202 and to improve response time for selected browsers. Traffic across the

TMP link 202 is reduced, for example, by compressing data. Compression can be implemented using any available compression mechanism and may operate on a packet-by-packet level or by assembling data from multiple packets to compress across a larger data set. Although compression may be applied equally to all data, it is known that some types of data do not benefit from compression. It is also known that certain compression mechanisms and algorithms are better suited for particular types of data. Accordingly, the present invention contemplates the dynamic selection of a compression mechanism based on the type of data being processed. For example, HTML data, which makes up a large proportion of web-based traffic, typically includes ASCII text which is known to compress well using, for example, compressed HTML mechanisms. Encrypted data, however, often does not compress well. Accordingly, the present invention may be implemented to apply compressed HTML techniques to HTML packets while passing encrypted packets (e.g., packets using a secure HTTP scheme) without attempting encryption. So long as front-end 201 and back-end 203 share a common semantic for performing the compression/decompression processes, any available algorithm may be implemented.

Encryption processes are largely analogous to compression processes in that they may be implemented by a number of available cipher algorithms and mechanisms including stream ciphers and block ciphers providing various levels of data security. It usually is not valuable to encrypt data that is already encrypted, hence it is contemplated that encryption may be selectively applied. Moreover, a vast majority of data transferred in many applications does not require encryption at all. The particular encryption mechanism used by the front-end 201 and back-end 203 can be selected based upon the type of

data, or designated on a file-by-file basis by a manager of server 210, for example. Front-end 201 and back-end 203 must share a common encryption/decryption semantic, however.

5 In one embodiment, front-end 201 and back-end 203 share operational information such as time synchronization and quality of service metrics with each other. This information is readily communicated by specially designated packets transmitted on TMP link 202, and/or by
10 including a portion of each TMP packet that is used to exchange this operational information. Traffic across link 202 is preferably managed by selectively transmitting packets at a rate determined to provide adequate quality of service and suitable packet delivery time using this
15 knowledge shared between the front-end 201 and back-end 203. Optionally, this operational information can be shared with processes running on client 205 and/or server 210 as well, although such sharing would require special configuration of client 205 and/or server 210 and is not
20 required to achieve the benefits of the present invention.

Traffic may be further reduced by using forward error correction (FEC) techniques to compensate for lossy connections. A variety of FEC techniques are known that add various amounts of overhead to the traffic. The
25 selection of a particular method depends on the quality of service (i.e., transit times and packet loss rate and/or bit error rate) of the communication channel being used. In one implementation, a statically defined FEC mechanism can be implemented between front-end 201 and back-end 203
30 based on average or worst-case quality of service (QoS). However, because both front-end 201 and back-end 203 have knowledge of the QoS metrics of each other and are time synchronized, it is contemplated that the FEC mechanisms

can be adaptive to current QoS metrics. For example, a data packets may be encoded with a 1-bit/byte error correction code during times of high QoS, and dynamically changed to a 3-bit/byte or 4-bit/byte error correction (or higher) encoding when QoS degrades. So long as front-end 201 and back-end 203 share a common semantic for handling the FEC processes, the actual implementation of those processes is very flexible and can be dynamically defined.

10 The blending of request datagrams results in fewer request:acknowledge pairs across the TMP link 202 as compared to the number required to send the packets individually between front-end 201 and back-end 203. This action reduces the overhead associated with transporting a
15 given amount of data, although conventional request:acknowledge traffic is still performed on the links coupling the front-end 201 to client 205 and back-end 203 to a web server. Moreover, resend traffic is significantly reduced further reducing the traffic.
20 Response time is further improved for select privileged users and for specially marked resources by determining the priority for each HTTP transmission.

In one embodiment, front-end 201 and back-end 203 are closely coupled to the Internet backbone. This means they
25 have high bandwidth connections, can expect fewer hops, and have more predictable packet transit time than could be expected from a general-purpose connection. Although it is preferable to have low latency connections between front-ends 201 and back-ends 203, a particular strength of
30 the present invention is its ability to deal with latency by enabling efficient transport and traffic prioritization. Hence, in other embodiments front-end 201 and/or back-end 203 may be located farther from the

Internet backbone and closer to clients 205 and/or web servers 210. Such an implementation reduces the number of hops required to reach a front-end 201 while increasing the number of hops within the TMP link 202 thereby
5 yielding control over more of the transport path to the management mechanisms of the present invention.

Clients 205 no longer conduct all data transactions directly with the web server 210. Instead, clients 205 conduct some and preferably a majority of transactions
10 with front-ends 201, which simulate the functions of web server 210. Client data is then sent, using TMP link 202, to the back-end 203 and then to the web server 210. Running multiple clients 205 over one large connection provides several advantages:

- 15 • Since all client data is mixed, each client can be assigned a priority. Higher priority clients, or clients requesting higher priority data, can be given preferential access to network resources so they receive access to the channel sooner while ensuring low-priority
20 clients receive sufficient service to meet their needs.
- The large connection between a front-end 201 and back-end 203 can be permanently maintained, shortening the many TCP/IP connection sequences normally required for many clients connecting and disconnecting.
- 25 • Services such as encryption, compression, error correction and time synchronization that may not be available or efficiently implemented in particular clients 205 can be practically implemented in TMP link where the resources required to provide these services
30 are shared across multiple clients 205.

Using a proprietary protocol allows the use of more effective techniques to improve data throughput and makes better use of existing bandwidth during periods when the network is congested.

5 A particular advantage of the architecture shown in FIG. 2 is that it is readily scaled. Any number of client machines 205 may be supported. In a similar manner, a web site owner may choose to implement a site using multiple web servers 210 that are co-located or distributed throughout network 101. To avoid congestion, additional front-ends 201 may be implemented or assigned to particular web sites. Each front-end 201 is dynamically re-configurable by updating address parameters to serve particular web sites. Client traffic is dynamically
10 directed to available front-ends 201 to provide load balancing. Hence, when quality of service drops because of a large number of client accesses, an additional front-end 201 can be assigned to the web site and subsequent client requests directed to the newly assigned front-end
15 201 to distribute traffic across a broader base.

In the particular examples, this is implemented by a front-end manager component 207 that communicates with multiple front-ends 201 to provide administrative and configuration information to front-ends 201. Each front-
25 end 201 includes data structures for storing the configuration information, including information identifying the IP addresses of web servers 210 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 may
30 include information for prioritizing data from and to particular clients, quality of service information, and the like.

Similarly, additional back-ends 203 can be assigned to a web site to handle increased traffic. Back-end manager component 209 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold current configuration state, quality of service information and the like. In the particular examples front-end manager 207 and back-end manager 209 serve multiple web sites 210 and so are able to manipulate the number of front-ends and back-ends assigned to each web site 210 by updating this configuration information. When the congestion for the site subsides, the front-end 201 and back-end 203 can be reassigned to other, busier web sites. These and similar modifications are equivalent to the specific examples illustrated herein.

In the case of web-based environments, front-end 201 is implemented using custom or off-the-shelf web server software. Front-end 201 is readily extended to support other, non-web-based protocols, however, and may support multiple protocols for varieties of client traffic. Front-end 201 processes the data traffic it receives, regardless of the protocol of that traffic, to a form suitable for transport by TMP 202 to a back-end 203. Hence, most of the functionality implemented by front-end 201 is independent of the protocol or format of the data received from a client 205. Hence, although the discussion of the exemplary embodiments herein relates primarily to front-end 201 implemented as a web server, it should be noted that, unless specified to the contrary, web-based traffic management and protocols are merely examples and not a limitation of the present invention.

As shown in FIG. 2, in accordance with the present invention a web site is implemented using an originating

web server 210 operating cooperatively with the web server of front-end 201. More generally, any network service (e.g., FTP, VoIP, NNTP, MIME, SMTP, Telnet, DBMS) can be implemented using a combination of an originating server
5 working cooperatively with a front-end 201 configured to provide a suitable interface (e.g., FTP , VoIP, NNTP, MIME, SMTP, Telnet, DBMS, WAP) for the desired service. In contrast to a simple front-end cache or proxy software, implementing a server in front-end 201 enables portions of
10 the web site (or other network service) to actually be implemented in and served from both locations. The actual web pages or service being delivered comprises a composite of the portions generated at each server. Significantly, however, the web server in front-end 201 is close to the
15 browser in a client 205 whereas the originating web server is close to all resources available at the web hosting center at which web site 210 is implemented. In essence the web site 210 is implemented by a tiered set of web servers comprising a front-end server 201 standing in
20 front of an originating web server.

This difference enables the web site or other network service to be implemented so as to take advantage of the unique topological position each entity has with respect to the client 205. By way of a particular example,
25 consider an environment in which the front-end server 201 is located at the location of an ISP used by a particular set of clients 205 and back-end 203 is closely coupled by a private channel to server 210. In such an environment, clients 205 can access the front-end server 205 without
30 actually traversing the network 101, hence the need for encryption and error correction and time synchronization services are relaxed with respect to the client-to-front-end link. In such cases the services provided transparently by enhanced channel 202 are substantially a

complete substitute for prior services implemented by modifying client 205 and server 210 themselves.

In order for a client 205 to obtain service from a front-end 201, it must first be directed to a front-end 201 that can provide the desired service. Preferably, client 205 does not need to be aware of the location of front-end 201, and initiates all transactions as if it were contacting the originating server 210. FIG. 3 illustrates a domain name server (DNS) redirection mechanism that illustrates how a client 205 is connected to a front-end 201. The DNS systems is defined in a variety of Internet Engineering Task Force (IETF) documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In a typical environment, a client 205 executes a browser 301, TCP/IP stack 303, and a resolver 305. For reasons of performance and packaging, browser 301, TCP/IP stack 303 and resolver 305 are often grouped together as routines within a single software product.

Browser 301 functions as a graphical user interface to implement user input/output (I/O) through monitor 311 and associated keyboard, mouse, or other user input device (not shown). Browser 301 is usually used as an interface for web-based applications, but may also be used as an interface for other applications such as email and network news, as well as special-purpose applications such as database access, telephony, and the like. Alternatively, a special-purpose user interface may be substituted for the more general-purpose browser 301 to handle a particular application.

TCP/IP stack 303 communicates with browser 301 to convert data between formats suitable for browser 301 and IP format suitable for Internet traffic. TCP/IP stack

also implements a TCP protocol that manages transmission of packets between client 205 and an Internet service provider (ISP) or equivalent access point. IP protocol requires that each data packet include, among other things, an IP address identifying a destination node. In current implementations the IP address comprises a 32-bit value that identifies a particular Internet node. Non-IP networks have similar node addressing mechanisms. To provide a more user-friendly addressing system, the Internet implements a system of domain name servers that map alpha-numeric domain names to specific IP addresses. This system enables a name space that is more consistent reference between nodes on the Internet and avoids the need for users to know network identifiers, addresses, routes and similar information in order to make a connection.

The domain name service is implemented as a distributed database managed by domain name servers (DNSs) such as DNS_A, DNS_B and DNS_C shown in FIG. 3. Each DNS relies on <domain name:IP> address mapping data stored in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files typically comprise text files that are read by a local name server, and hence become available through the name servers 307 to users of the domain system.

The user programs (e.g., clients 205) access name servers through standard programs such as resolver 305. Resolver 305 includes an address of a DNS 307 that serves as a primary name server. When presented with a reference to a domain name (e.g., <http://www.circadence.com>), resolver 305 sends a request to the primary DNS (e.g., DNS_A in FIG. 3). The primary DNS 307 returns either the

IP address mapped to that domain name, a reference to another DNS 307 which has the mapping information (e.g., DNS_B in FIG. 3), or a partial IP address together with a reference to another DNS that has more IP address
5 information. Any number of DNS-to-DNS references may be required to completely determine the IP address mapping.

In this manner, the resolver 305 becomes aware of the IP address mapping which is supplied to TCP/IP component 303. Client 205 may cache the IP address mapping for
10 future use. TCP/IP component 303 uses the mapping to supply the correct IP address in packets directed to a particular domain name so that reference to the DNS system need only occur once.

In accordance with the present invention, at least
15 one DNS server 307 is owned and controlled by system components of the present invention. When a user accesses a network resource (e.g., a web site), browser 301 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional
20 manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210 and server 210 performs a redirection of the browser to the system owned DNS server (i.e., DNS_C in FIG. 3). In a second embodiment,
25 domain:address mappings within the DNS system are modified such that resolution of the of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of
30 further redirecting the browser 301 to the best available front-end 201.

Unlike a conventional DNS server, however, the system-owned DNS_C in FIG. 3 receives domain:address

mapping information from a redirector component 309. Redirector 309 is in communication with front-end manager 207 and back-end manager 209 to obtain information on current front-end and back-end assignments to a particular server 210. A conventional DNS is intended to be updated infrequently by reference to its associated master file. In contrast, the master file associated with DNS_C is dynamically updated by redirector 309 to reflect current assignment of front-end 201 and back-end 203. In operation, a reference to web server 210 (e.g., http://www.circadance.com) may result in an IP address returned from DNS_C that points to any selected front-end 201 that is currently assigned to web site 210. Likewise, web site 210 may identify a currently assigned back-end 203 by direct or indirect reference to DNS_C.

Front-end 201 typically receives information directly from front-end manager 207 about the address of currently assigned back-ends 203. Similarly, back-end 203 is aware of the address of a front-end 201 associated with each data packet. Hence, reference to the domain system is not required to map a front-end 201 to its appropriate back-end 203.

FIG. 4 illustrates principle functional components of an exemplary front-end 201 in greater detail. Primary functions of the front-end 201 include translating transmission control protocol (TCP) packets from client 205 into TMP packets used in the system in accordance with the present invention. It is contemplated that various functions described in reference to the specific examples may be implemented using a variety of data structures and programs operating at any location in a distributed network. For example, a front-end 201 may be operated on

a network appliance 107 or server within a particular network 102, 103, or 104 shown in FIG. 1.

TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792, RFC1112 that are incorporated by reference herein. For ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement file transfer protocol (FTP), mail protocols such as

simple mail transport protocol (SMTP), structured query language (SQL) and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

To improve performance, front-end 201 optionally includes a caching mechanism 403. Cache 403 may be implemented as a passive cache that stores frequently and/or recently accessed web pages or as an active cache that stores network resources that are anticipated to be accessed. In non-web applications, cache 403 may be used to store any form of data representing database contents, files, program code, and other information. Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for data within cache 403. If the request can be satisfied from cache 403, the data is supplied directly without reference to web server 210 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by web sever 210. Also, cache 403 may proactively update the cache contents even before a request is received for particularly important or frequently used data from web server 210. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out, or random eviction. When the requested data is not within cache 403, a request is processed to web server 210, and the returned data may be stored in cache 403.

Several types of packets will cause parser 404 to forward a request towards web server 210. For example, a request for data that is not within cache 403 (or if

optional cache 403 is not implemented) will require a reference to web server 210. Some packets will comprise data that must be supplied to web server 210 (e.g., customer credit information, form data and the like). In these instances, HTTP parser 402 couples to data blender 404.

In accordance with the present invention, front-end 201 implements security processes, compression processes, encryption processes, error correction processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within pre-processing unit 408, or alternatively implemented within any of the functional components within front-end 201. Also, front-end 201 may implement a prioritization program to identify packets that should be given higher priority service. A prioritization program requires only that front-end 201 include a data structure associating particular clients 205 or particular TCP packet types or contents with a prioritization value. Based on the prioritization value, parser 402 may selectively implement such features as caching, encryption, security, compression, error correction and the like to improve performance and/or functionality. The prioritization value is provided by the owners of web site 210, for example, and may be dynamically altered, statically set, or updated from time to time to meet the needs of a particular application.

Blender 404 slices and/or coalesces the data portions of the received packets into a more desirable "TMP units" that are sized for transport through the TMP mechanism 212. The data portion of TCP packets may range in size depending on client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where

compression is applied, the compressed data will vary in size depending on the compressibility of the data. Data blender 404 receives information from front-end manager 217 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism 212. Data blender 404 also marks the TMP units so that they can be re-assembled at the receiving end. Data blender 404 may also serve as a buffer for storing packets from all appliances 107 that are associated with front-end 201. In accordance with the present invention, data blender 404 may associate a prioritization value with each packet.

TMP mechanism implements a TMP protocol, described in greater detail hereinbelow, to communicate TMP packets. Received TMP packets include subpackets from multiple TCP connections. The data portions of subpackets are reassembled by reassemble mechanism 406 into a form suitable for return to the requesting client 205. For example, in an HTTP environment reassemble mechanism 406 creates HTTP response payloads akin to what would have been generated by an origin server 210.

Postprocessing mechanism 407 performs decompression, decryption, forward error correction and the like on packets received from a back-end 203. As described hereinafter with respect to FIG. 5, back-end 203 preferably includes pre-processing mechanisms 508 that are analogous to pre-processing mechanisms 408. Hence, post-processing mechanisms 407 restore the data to a form usable by a client 205 without additional processing. Accordingly, client 205 need not implement any of the pre-processing or post processing functions while still realizing the benefits of these processes.

FIG. 5 illustrates principle functional components of an exemplary back-end 203 in greater detail. Primary functions of the back-end 203 include translating transmission control protocol (TCP) packets from web server 210 into TMP packets as well as translating TMP packets received from a front-end 201 into the one or more corresponding TCP packets to be send to server 210. Further, back-end 203 is able to implement similar or complementary functionality to that of front-end 203. In this manner, back-end 203 can operate as a web server to retrieve content and generate web pages, analyze and reformat web pages and components within web pages, and similar server functionality that would conventionally be implemented in a server 210. In general, any functionality and behavior described herein that can be implemented on server 210 and/or front-end server 201 can also be implemented on back-end server 203.

TMP unit 505 receives TMP packets from TMP pipe 212 and passes them to HTTP reassemble unit 507 where they are reassembled into the corresponding TCP packets. Data filter 506 may implement other functionality such as decompression, decryption, and the like to meet the needs of a particular application. The reassembled data is forwarded to TCP component 501 for communication with web server 210.

TCP data generated by the web server process are transmitted to TCP component 501 and forwarded to HTTP parse mechanism 502. Parser 502 operates in a manner analogous to parser 402 shown in FIG. 5 to extract the data portion from the received TCP packets. Pre-processing mechanism 508 and post-processing mechanism 507 operate in an analogous fashion to components 407 and 408 to perform compression, encryption, error correction, and

the like, and forward those packets to data blender 504. Data blender 504 operates in a manner akin to data blender 404 shown in FIG. 5 to buffer and prioritize packets in a manner that is efficient for TMP transfer. Priority information is received by, for example, back-end manager 209 based upon criteria established by the web site owner. TMP data is streamed into TMP unit 505 for communication on TMP pipe 212.

In an exemplary implementation, illustrated in FIG. 6 and FIG. 7, a "TMP connection" comprises a plurality of "TCP connection buffers", logically arranged in multiple "rings". Each TCP socket 701 maintained between the front-end 201 and a client 205 corresponds to a TCP connection buffer 702. Pre-processing 408 is performed on the TCP connection buffer data to provide, for example, data compression, encryption, and/or error correction coding before the data is placed in the corresponding TCP connection buffer 702.

When a TCP connection buffer 702 is created, it is assigned a priority. For purposes of the present invention, any algorithm or criteria may be used to assign a priority. Each priority ring is associated with a number of TCP connection buffers having similar priority. In a specific example, five priority levels are defined corresponding to five priority rings. Each priority ring is characterized by the number of connection buffers it holds (nSockets), the number of connection buffers it holds that have data waiting to be sent (nReady) and the total number of bytes of data in all the connection buffers that it holds (nBytes).

A TCP connection buffer 702 is created and placing one or more preprocessed packets from a TCP socket 701 within the TCP connection buffer 702. A TCP connection

buffer 702 is sized to hold a plurality of TCP packets and each TCP connection buffer 702 is associated with a priority value. The priority value is assigned when TCP connection buffer 702 is first created and may be dynamically changed in operation.

When sending data, blender 404 performs a series of processes outlined in FIG. 6 that access data from the TCP connection buffers 702 to form TMP units 705 that are transmitted. The processes performed by blender 404 include:

In step 602, determine the number of bytes available to be sent from each ring (nBytes), and the number of TCP connections that are ready to send (nReady)

In step 603, determine how many bytes should be sent from each ring. This is based on a weight parameter for each priority. The weight can be thought of as the number of bytes that should be sent at each priority this time through the loop.

The nSend value computed in the previous step 603 reflects the weighted proportion that each ring will have in a blended TMP packet, but the values of nSend do not reflect how many bytes need to be selected to actually empty most or all of the data waiting to be sent a single round. To do this, the nSend value is normalized to the ring having the most data waiting (e.g., nBytes = nSendNorm) in step 604. This involves a calculation of a factor: $S = nBytes / (Weight * nReady)$ for the ring with the greatest nReady. Then, for each ring, calculate $nReady * S * Weight$ to get the normalized value (nSendNorm) for each priority ring.

In step 605, sub-packets are sent from the different rings. This is done, for example, by taking a sub-packet from the highest priority ring and adding it to a TMP packet, then adding a sub-packet from each of the top two queues, then the top three, and so on. A variety of algorithms may be used to select particular sub-packets from the different rings to implement a desired level of fairness, prioritization, and quality of service.

Referring to step 606, within each ring, sub-packets are added round robin. When a sub-packet is added from a TCP connection buffer the ring is rotated so the next sub-packet the ring adds will come from a different TCP connection buffer. Each sub-packet can be up to 512 bytes in a particular example. If the connection buffer has less than 512 bytes waiting, the data available is added to the TMP packet.

In step 607, when a full TMP packet (roughly 1.5 kB in a particular example) is built, it is sent. This can have three or more sub packets, depending on their size. The TMP packet will also be sent when there is no more data ready.

TMP unit 405 (shown in FIG. 4) and TMP unit 505 (shown in FIG. 5) implement the TMP protocol that communicates packets between front-end 201 and back-end 203. The protocol is rides on top of universal datagram protocol (UDP) in that network devices that handle TMP packets treat them as UDP packets. However, TMP packets differ from standard UDP packets in that they have additional unique header data defining a unique set of messages, outlined below, to support the TMP functionality. Also, the manner in which TMP packets are transferred onto the physical communication channel,

referred to as the protocol behavior, differs significantly from TCP.

TMP packets have a header that contains packet control information. Some TMP packets also carry extra
5 information in a data or payload portion. The packet control information includes, for example:

- A connection number (that identifies the connection to which it belongs)
- A checksum for data integrity
- 10 • A set of flags (which may be used or remain unused) for a variety of purposes
- A message type identifier
- The confirmed message type

The rest of the packet header contains information or data
15 which can differ between packets, depending on the message type.

A short list of messages that can be sent by the TMP protocol includes: data, acknowledgments, connection requests and replies, time synchronization requests and
20 replies, resent data, control messages, QoS messages, status requests and replies, suspend messages, and alerts. Packet header content which is specific to the message type is as follows.

- Acknowledgment
 - 25 o The last sequential confirmed sequence message
 - o The confirmed message sequence number
- Time Synchronization Request
 - o Requester time index.
- Time Synchronization Reply
 - 30 o The time that the request was received.

- o The time that the reply was sent.
 - o Requester time index.
- Connection Request
 - o The connections index (zero for a new connection).
 - o Requested receiving port.
 - o An additional set of flags (which may be used or unused) for a variety of purposes.
- Connection Reply
 - o The replier's base time.
 - o A time offset from the point of receiving the request in milliseconds.
 - o The connections index (zero for a new connection).
 - o An additional set of flags (which may be used or unused) for a variety of purposes.
- Data
 - o Data sequence number.
 - o Time that the message was sent.

The rest of the packet comprises the packet body or payload portion. Alert and Acknowledge packets do not have bodies. All other packets contain bodies that carry additional information appropriate to the message itself (for example, a data packet will send the data itself).

It is important to note that alerts and QoS information are built into the protocol and do not need to be passed as data packets. Since these types of information are not built into TCP they would need to be sent as data, which might affect the application using the protocol. This means that the receiving end needs to process the packet only once to draw out the information it requires. In contrast, when QoS information is sent as a data packet in TCP, the receiving end has to process the

packet as a data packet simply to get to the information that allows the alert or QoS information to be processed, which means that TCP must double the amount of processing for alerts and QoS information.

5 Of particular interest in the present invention, the exchange of time synchronization information 707 enables front-end 201 and back-end 203 to have a common time base and ascertain the time of issue of any received packet. While the current implementation does not include base
10 time or time index data in the header of data packets, this information can readily be included in all message types, a subset of message types, and/or in a special message type defined for real-time data transport. In this manner, the recipient of a TMP packet knows with a
15 high level of certainty when a received packet was transmitted, something that existing Internet protocols do not provide. In the case of TMP packets from a back-end 203 to a front-end 201, the information can be used by the front-end 201 as a factor in ordering responses to clients
20 205. In the case of TMP packets from a back-end 203 to a front-end 201, the information can be used by the front-end 203 as a factor in ordering responses to clients 205.

 Rather than synchronizing clocks the front-end 201 and back-end 203 (i.e., absolute time synchronization),
25 the time synchronization information 707 may indicate a differential between the clocks of the two machines (i.e., relative time synchronization). Relative time synchronization can be used substantially equivalently to information that would allow actual synchronization of the
30 clocks. Accordingly, "time synchronization" and "time synchronized" refer inclusively to both absolute and relative time synchronization methods.

The time synchronization information 707 augments or replaces the "time to live" feature of conventional IP packets. Each IP packet specifies a time to live value that must be decremented by each router or device that handles the packet. As the time value can only be incremented in one-second units, the value becomes a hop count rather than an actual timing function. When a packet's time to live value is decremented to zero, it is discarded and must be retransmitted. In accordance with the present invention, the time to live value for TMP packets can be used more meaningfully as the recipient knows when the packet was actually sent and can set or reset the time to live value to a meaningful value when the packet leaves a front-end 201 or back-end 203.

As in all protocols, the messages in TMP have an order in which they are sent as well as particular defined situations in which they are sent. A typical TMP session might begin with a connection request. For reference, the end point that sends the connection request will be referred to as the front-end, and the receiver of the request will be referred to as the back-end, although the TMP protocol operates bi-directionally between front-ends and back-ends. The front-end 201 sends a connection request to the back-end 203, and the back-end 203 sends a connection reply back to the front-end 201. This reply will be either positive (connection accepted), or negative (connection refused). If the reply is positive, then the connection is established and the front-end and back-end can begin to exchange data.

TMP is a TCP-like protocol adapted to improve performance for multiple connections operating over a single pipe. The TMP mechanism in accordance with the present invention creates and maintains a stable

connection between two processes for high-speed, reliable, adaptable communication. TMP is not merely a substitute for the standard TCP environment. TMP is designed to perform particularly well in heterogeneous network environments such as the Internet. TMP connections are made less often than TCP connections. Once a TMP connection is made, it remains up unless there is some kind of direct intervention by an administrator or there is some form of connection-breaking network error. This reduces overhead associated with setting up, maintaining and tearing down connections normally associated with TCP.

Another feature of TMP is its ability to channel numerous TCP connections through a single TMP pipe 202. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then mapped to a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 202 in order to traffic the TCP connections to their appropriate destinations. TMP includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

Another advantage of TMP as compared to traditional protocols is the amount of information about the quality of the connection that a TMP connection conveys from one end to the other of a TMP pipe 202. As often happens in a network environment, each end has a great deal of information about the characteristics of the connection in one direction, but not the other. QoS information 708 is exchanged between front-end 201 and back-end 203 in accordance with the present invention. By knowing about the connection as a whole, TMP can better take advantage of the available bandwidth.

A QoS message is sent alone or may be piggybacked on a data packet. It sends information regarding the connection from one end of the connection to the other. Both front-end 201 and back-end 203 send QoS messages.

5 The information in a QoS message is the most up to date that the sending end has. That means that if a QoS message is to be resent, the QoS information is updated before it is resent. A QoS message is identified by the message type flag QoS. In a particular implementation, a
10 QoS message contains:

- 16 Bits - Average round trip time (RTT). This indicates the average round trip time as calculated by this end of the system over the last time interval, measured in milliseconds.

- 15
- 32 Bits - Packets Sent. This indicates the number of packets that were sent in the last time interval.

- 20
- 32 Bits - Packets Received. This indicates the number of packets that were received in the last time interval.

- 32 Bits - Packets Resent. This indicates the number of packets that needed to be resent in the last time interval.

- 25
- 16 Bits - Window Size. This value indicates the current window size that one end is operating under. This will allow for a random sampling of window sizes to be gathered at the other end.

- 30
- 16 Bits - Packets in Flight. This value indicates the current number of packets that one end has sent to the other end without receiving an

acknowledgement. This will allow for a random sampling of packets in flight to be gathered by the other end.

- 32 Bits - Time Interval. The span of time that the information in the QoS packet is dealing with. This parameter is measured in seconds.

In this manner, both front-end 201 and back-end 203 are aware of not only their own QoS metrics, but also those of the machine with which they are communicating and their shared communication link.

As suggested in FIG. 7, QoS information 708 and time synchronization information 707 can be used by blender 404 to select the order in which data is placed into TMP units 705. Also, QoS information 708 can be used by TMP mechanisms 405 and 505 to alter the TMP behavior.

In contrast with conventional TCP mechanisms, the behavior implemented by TMP mechanism 405 is constantly changing. Because TMP obtains bandwidth to host a variable number of TCP connections and because TMP is responsive to information about the variable status of the network, the behavior of TMP is preferably continuously variable. One of the primary functions of TMP is being able to act as a conduit for multiple TCP connections. As such, a single TMP connection cannot behave in the same manner as a single TCP connection. For example, imagine that a TMP connection is carrying 100 TCP connections. At this time, it loses one packet. TCP would require that the connection bandwidth be cut in half. This is a performance reduction on 100 connections instead of just on the one that lost the packet.

Each TCP connection that is passed through the TMP connection must get a fair share of the bandwidth, and should not be easily squeezed out by competing users of the available bandwidth. To allow this to happen, every

5 TMP connection becomes more aggressive in claiming bandwidth as it accelerates. Like TCP, the bandwidth available to a particular TMP connection is measured by its window size (i.e., the number of outstanding TCP packets that have not yet been acknowledged). Bandwidth

10 is increased by increasing the window size, and relinquished by reducing the window size. Up to protocol specified limits, each time a packet is successfully delivered and acknowledged, the window size is increased until the window size reaches a protocol specified

15 maximum. When a packet is dropped (e.g., no acknowledge received or a resend packet response is received), the bandwidth is decreased by backing off the window size. TMP also ensures that it becomes more and more resistant to backing off (as compared to TCP) with each new TCP

20 connection that it hosts. Further, a TMP should not go down to a window size of less than the number of TCP connections that it is hosting.

In a particular implementation, every time a TCP connection is added to (or removed from) what is being

25 passed through the TMP connection, the TMP connection behavior is altered. It is this adaptation that ensures successful connections using TMP. Through the use of the adaptive algorithms discussed above, TMP is able to adapt the amount of bandwidth that it uses. When a new TCP

30 connection is added to the TMP connection, the TMP connection becomes more aggressive to accommodate it. When a TCP connection is removed from the TMP connection, the TMP connection becomes less aggressive.

TMP connection 202 provides improved performance in its environment as compared to conventional TCP channels, but it is recognized that TMP 202 resides on the Internet in the preferred implementations. Hence, TMP must live
5 together with many protocols and share the pipe efficiently in order to allow the other transport mechanisms fair access to the shared communication bandwidth. Since TMP takes only the amount of bandwidth that is appropriate for the number of TCP connections that
10 it is hosting (and since it monitors the connection and controls the number of packets that it puts on the line), TMP will exist cooperatively with TCP traffic. Furthermore, since TMP does a better job at connection monitoring than TCP, TMP is better suited to throughput
15 and bandwidth management than TCP.

FIG. 8 illustrates an exemplary set of processes 808 implemented by pre-processing units 408 and 508. Some, none, or all processes illustrated in FIG. 8 may be implemented on particular packets as described
20 hereinbefore. Unprocessed payload 801 from a payload portion of a packet are passed to processes 808 that perform encryption, compression, and/or error correction. The actual algorithms used to implement encryption, compression and/or error correction in any specific
25 implementation are a design choice made be to meet the needs of a particular application. Error correction is preferably forward error correction that adds redundant data to the pre-processed payload so that a recipient can reconstruct the payload portion in the presence of one or
30 more transmission errors. The amount and format of redundant information can be varied dynamically to account for current QoS conditions as reported by, for example, QoS information 708.

FIG. 9 illustrates an exemplary set of processes implemented by post-processing units 407 and 507. Some, none, or all processes illustrated in FIG. 9 may be implemented on particular packets depending on the corresponding pre-processing performed on the packets. Pre-processed packets are passed to processes that perform decryption, decompression, and/or error correction decoding. The actual algorithms used in any specific implementation are determined to complement the pre-processing processes. Error correction operates to detect one or more transmission errors, determine if the detected errors are correctable, and when correctable, reforming the corrected payload. Payload portion 903 is essentially a fully-formed payload portion of, for example, an HTTP packet.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols and languages including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back-end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols and languages such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end supports a DBMS protocol such as SQL. Such implementations not only provide the advantages of the present invention, but also enable a client to

access a rich set of network resources with minimal client software.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100